Compositional Event Structure Semantics of the Internal pi-Calculus

Silvia Crafa, Daniele Varacca, Nobuko Yoshida

Compositional Event Structure Semantics for the π-Calculus

Silvia Crafa[1] Daniele Varacca[2] Nobuko Yoshida[3]

[1]Università di Padova [2] PPS - Université Paris 7 & CNRS [3]Imperial College London

Abstract. We propose the first compositional event structure semantics for a fully expressive π-calculus, generalising Winskel's event structures for CCS. The π-calculus we model is the πI-calculus with recursive definitions and summations. First we model the *synchronous* calculus, introducing a notion of dynamic renaming to the standard operators on event structures. Then we model the *asynchronous* calculus, for which a new additional operator, called *rooting*, is necessary for representing causality due to new name binding. The semantics are shown to be operationally adequate and sound with respect to bisimulation.

1 Introduction

Event structures [17] are a causal model for concurrency which is particularly suited for the traditional process calculi such as CCS, CSP, SCCS and ACP. Event structures intuitively and faithfully represent *causality* and *concurrency*, simply as a partial order and an irreflexive binary relation. The key point of the generality and applicability of this model is the compositionality of the parallel composition operator: the behaviour of the parallel composition of two event structures is determined by the behaviours of the two event structures. This modularity, together with other algebraic operators such as summation, renaming and hiding, leads also to a straightforward correspondence between the event structures semantics and the operational semantics - such as the labelled transition system - of a given calculus [25].

In this paper we propose the first compositional event structure semantics of a fully expressive variant of the π-calculus. The semantics we propose generalises Winskel's semantics of CCS [21], it is operationally adequate with respect to the standard labelled transition semantics, and consequently it is sound with respect to bisimilarity.

The π-calculus we consider is known in the literature as the πI-calculus [18], where the output of free names is disallowed. The symmetry of input and output prefixes, that are both binders, simplifies considerably the theory, while preserving the basic expressiveness of the calculi with free name passing [2, 16].

In order to provide an event structure semantics of the π-calculus, one has in particular to be able to represent dynamic creations of new synchronisation channels, a feature that is not present in traditional process algebras. In Winskel's event structure semantics of CCS [21], the parallel composition is defined as product in a suitable category followed by relabelling and hiding. The product represents all conceivable synchronisations, the hiding removes synchronisations that are not allowed, while the relabelling chooses suitable names for synchronisation events. In CCS one can decide statically whether two events are allowed to synchronise, whereas in the π-calculus, a synchronisation between two events may depend on which synchronisations took place before.

Consider for instance the π-process $a(x).\overline{x}(u).\mathbf{0} \mid \overline{a}(z).z(v).\mathbf{0}$ where $a(x).P$ is an input at a, $\overline{a}(z).Q$ is an output of a new name z to a and $\mathbf{0}$ denotes the inaction. This process contains two synchronisations, first along the channel a and then along a private, newly created, channel z. The second synchronisation is possible only since the names x and z are made equal by the previous synchronisation along a. To account for this phenomenon, we define the semantics of the parallel composition by performing hiding and relabelling not uniformly on the whole event structure, but relative to the causal history of events.

The full symmetry underlying the πI-calculus theory has a further advantage: it allows a uniform treatment of causal dependencies. Causal dependencies in the π-processes arise in two ways [3, 10]: by nesting prefixes (called *structural* or *prefixing* or *subject* causality) and by using a name that has been bound by a previous action (called *link* or *name* or *object* causality). While subject causality is already present in CCS, object causality is distinctive of the π-calculus. In the synchronous πI-calculus, object causality always appear under subject causality, as in $a(x).x(y).\mathbf{0}$ or in $(\nu c)(a(x).c(z).\mathbf{0} \mid \overline{c}(w).x(y).\mathbf{0})$, where the input on x causally depends in both senses from the input on a. As a result, the causality of synchronous πI-calculus can be naturally captured by the standard prefixing operator of the event structures, as in CCS.

On the other hand, in the asynchronous πI-calculus, the bound output process is no longer a prefix: in $\overline{a}(x)P$, the continuation process P can perform any action α before the output of x on a, provided that α does not contain x. Thus the asynchronous output has a looser causal dependency. For example, in $(\nu c)(\overline{a}(x)c(z).\mathbf{0} \mid \overline{c}(w)x(y).\mathbf{0})$, $\overline{a}(x)$ only binds the input at x, and the interaction between $c(z)$ and $\overline{c}(w)$ can perform before $\overline{a}(x)$, thus there exists no subject causality. Representing this output object causality requires a novel operator on event structures that we call *rooting*, whose construction is inspired from a recent study on Ludics [8].

With these new constructions, the semantics of both the synchronous and the asynchronous πI-calculus is compositional, operationally adequate and sound with respect to bisimilarity.

2 Internal π-calculus

This section gives basic definitions of the πI-calculus [18]. This subcalculus captures the essence of name passing with a simple labelled transition relation. In contrast with the full π-calculus, only one notion of strong bisimulation exists, and it is a congruence.

2.1 Syntax

The syntax of the monadic, synchronous πI-calculus [18] is the following, where the symbols $a, b, \ldots, x, y, z$ range over the infinite set of names denoted by *Names*.

$$\begin{array}{lll} \textit{Prefixes} & & \pi ::= a(x) \mid \overline{a}(x) \\ \textit{Processes} & & P, Q ::= \sum_{i \in I} \pi_i.P_i \mid P \mid Q \mid (\nu a)P \mid A\langle \tilde{x} \mid \mathbf{z} \rangle \\ \textit{Definitions} & & A(\tilde{x} \mid \mathbf{z}) = P_A \end{array}$$

The syntax consists of the parallel composition, name restriction, finite summation of guarded processes and recursive definition. In $\sum_{i \in I} \pi_i.P_i$, I is a finite indexing set; when

I is empty we simply write $\mathbf{0}$ and denote with $+$ the binary sum. The two prefixes $a(x)$ and $\overline{a}(x)$ represent, respectively, an input prefix and a bound output prefix. A process $a(x).P$ can perform an input at a and x is the placeholder for the name so received. The bound output case is symmetric: a process $\overline{a}(x).P$ can perform an output of the fresh name x along the channel a. Differently from the π-calculus, where both bound and free names can be sent along channels, in the πI-calculus only bound names can be communicated, modelling the so called *internal mobility*. We often omit $\mathbf{0}$ and objects (e.g. write $\overline{a}$ instead of $\overline{a}(x).\mathbf{0}$).

The choice of recursive definitions rather than replication for infinite processes is justified by the fact that the πI-calculus with replication is strictly less expressive [18]. We assume that every constant A has a unique defining equation $A(\tilde{x} \mid \mathbf{z}) = P_A$. The symbol $\tilde{x}$ denotes a tuple of distinct names, while $\mathbf{z}$ represents an infinite sequence of distinct names $\mathbb{N} \to \mathit{Names}$. We denote $\mathbf{z}(n)$ as z_n. The tuple $\tilde{x}$ contains all free names of P_A and the range of $\mathbf{z}$ contains all bound names of P_A. The parameter $\mathbf{z}$ does not usually appear in recursive definitions in the literature. The reason we add it is that we want to maintain the following assumption:

Every bound name is different from any other name, either bound or free. (1)

In the π-calculus, this policy is usually implicit and maintained along the computation by dynamic α-conversion: every time the definition A is unfolded, a new copy of the process P_A is created whose bound names must be fresh. This dynamic choice of names is difficult to interpret in the event structures. Hence our recursive definitions prescribe all the names that will be possibly used for a precise semantic correspondence.

The set of free and bound names of P, written by $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$, is defined as usual, for instance $\mathrm{fn}(\overline{a}(x).P) = \{a\} \cup (\mathrm{fn}(P) \setminus \{x\})$. As for constant processes, the definition is as follows: $\mathrm{fn}(A\langle \tilde{x} \mid \mathbf{z} \rangle) = \{\tilde{x}\}$ and $\mathrm{bn}(A\langle \tilde{x} \mid \mathbf{z} \rangle) = \mathbf{z}(\mathbb{N})$.

2.2 Operational Semantics

The operational semantics of the calculus is given in terms of an LTS (in late style), which is defined as follows, where we let α, β range over the set of labels $\{\tau, a(x), \overline{a}(x)\}$.

(IN LATE) $\dfrac{}{a(x).P \xrightarrow{a(x)} P}$ (OUT) $\dfrac{}{\overline{a}(x).P \xrightarrow{\overline{a}(x)} P}$ (COMM) $\dfrac{P \xrightarrow{a(x)} P' \quad Q \xrightarrow{\overline{a}(y)} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)(P'\{y/x\} \mid Q')}$

(PAR) $\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$ (SUM) $\dfrac{P_i \xrightarrow{\alpha} P_i'}{\sum_{i \in I} P_i \xrightarrow{\alpha} P_i'}\ i \in I$ (RES) $\dfrac{P \xrightarrow{\alpha} P'}{(\nu a)P \xrightarrow{\alpha} (\nu a)P'}\ a \notin \mathrm{fn}(\alpha)$

(REC) $\dfrac{P_A\{\tilde{y}/\tilde{x}\}\{\mathbf{w}/\mathbf{z}\} \xrightarrow{\alpha} P'}{A\langle \tilde{y} \mid \mathbf{w} \rangle \xrightarrow{\alpha} P'}\ A(\tilde{x} \mid \mathbf{z}) = P_A$

The rules above illustrate the internal mobility characterising the πI-calculus communication. In particular, according to (COMM), we have that $a(x).P \mid \overline{a}(y).Q \xrightarrow{\tau} (\nu y)(P\{y/x\} \mid Q)$ where the fresh name y appearing in the output is chosen as the "canonical representative" of the private value that has been communicated. In (REC), the unfolding of a new copy of the recursive process updates the sequence of bound names. The formal definition of the substitution $\{\mathbf{w}/\mathbf{z}\}$ is found in Appendix A.1.

Proposition 1. *Let P be a process that satisfies Assumption 1. Suppose $P \xrightarrow{\alpha} P'$. Then P' satisfies Assumption 1.*

Example 1. Consider $A(x \mid \mathbf{z}) = x(z_0).A\langle z_0 \mid \mathbf{z}'\rangle \mid x(z_1).A\langle z_1 \mid \mathbf{z}''\rangle$, where $\mathbf{z}'(n) = \mathbf{z}(2n+2)$ and $\mathbf{z}''(n) = \mathbf{z}(2n+3)$. In this case the sequence of names $\mathbf{z}$ is partitioned into two infinite subsequences $\mathbf{z}'$ and $\mathbf{z}''$ (corresponding to even and odd name occurrences), so that the bound names used in the left branch of A are different from those used in the right branch. Intuitively $A\langle a \mid \mathbf{z}\rangle$ partially "unfolds" to $a(z_0).(z_0(z_2).A\langle z_2 \mid \mathbf{z}'_1\rangle \mid z_0(z_4).A\langle z_4 \mid \mathbf{z}'_2\rangle) \mid a(z_1).(z_1(z_3).A\langle z_3 \mid \mathbf{z}''_1\rangle \mid z_1(z_5).A\langle z_3 \mid \mathbf{z}''_2\rangle)$ with suitable $\mathbf{z}'_1, \mathbf{z}'_2, \mathbf{z}''_1, \mathbf{z}''_2$.

We end this section with the definition of strong bisimilarity in the πI-calculus.

Definition 1 (πI strong bisimilarity). *A symmetric relation $\mathcal{R}$ on πI processes is a strong bisimulation if $P\ \mathcal{R}\ Q$ implies:*

- *whenever $P \xrightarrow{\tau} P'$, there is Q' s.t. $Q \xrightarrow{\tau} Q'$ and $P'\mathcal{R}Q'$.*
- *whenever $P \xrightarrow{a(x)} P'$, there is Q' s.t. $Q \xrightarrow{a(y)} Q'$ and $P'\{z/x\}\mathcal{R}Q'\{z/y\}$.*
- *whenever $P \xrightarrow{\overline{a}(x)} P'$, there is Q' s.t. $Q \xrightarrow{\overline{a}(y)} Q'$ and $P'\{z/x\}\mathcal{R}Q'\{z/y\}$.*

with z being any fresh variable. Two processes P, Q are bisimilar*, written $P \sim Q$, if they are related by some strong bisimulation.*

This definition differs from the corresponding definition in [18] because we do not have the α-conversion rule, and thus we must allow Q to mimic P using a different bound name. The relation $\sim$ is a congruence and contains α-equivalence.

3 Event Structures

This section reviews basic definitions of event structures, that will be useful in Section 4. Event structures appear in the literature in different forms, the one we introduce here is usually referred to as prime event structures [9, 17, 22].

3.1 Basic definitions

Definition 2 (Event Structure). *An event structure is a triple $\mathcal{E} = \langle E, \leq, \smile\rangle$ s.t.*

- *E is a countable set of* events*;*
- *$\langle E, \leq\rangle$ is a partial order, called the* causal order*;*
- *for every $e \in E$, the set $\lceil e\rceil := \{e' \mid e' < e\}$, called the* enabling set *of e, is finite;*
- *$\smile$ is an irreflexive and symmetric relation, called the* conflict relation*, satisfying the following: for every $e_1, e_2, e_3 \in E$ if $e_1 \leq e_2$ and $e_1 \smile e_3$ then $e_2 \smile e_3$.*

The reflexive closure of conflict is denoted by $\asymp$. We say that the conflict $e_2 \smile e_3$ is *inherited* from the conflict $e_1 \smile e_3$, when $e_1 < e_2$. If a conflict $e_1 \smile e_2$ is not inherited from any other conflict we say that it is *immediate*. If two events are not causally related nor in conflict they are said to be *concurrent*.

Definition 3 (Labelled event structure). *Let L be a set of labels. A labelled event structure $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ is an event structure together with a labelling function $\lambda : E \to L$ that associates a label to each event in E.*

Intuitively, labels represent *actions*, and events should be thought of as *occurrences of actions*. Labels allow us to identify events which represent different occurrences of the same action. In addition, labels are essential when composing two event structures in a parallel composition, in that they are used to point out which events may synchronise.

In order to give the semantics of a process P as an event structure $\mathcal{E}$, we have to show how the computational steps of P are reflected into $\mathcal{E}$. This will be formalised in the Operational Adequacy Theorem 2 in Section 4, which is based on the following labelled transition systems over event structures.

Definition 4. *Let $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ be a labelled event structure and let e be one of its minimal events. The event structure $\mathcal{E} \lfloor e = \langle E', \leq', \smile', \lambda' \rangle$ is defined by: $E' = \{e' \in E \mid e' \not\asymp e\}$, $\leq' = \leq_{|E'}$, $\smile' = \smile_{|E'}$, and $\lambda' = \lambda_{E'}$. If $\lambda(e) = \beta$, we write $\mathcal{E} \xrightarrow{\beta} \mathcal{E} \lfloor e$.*

Roughly speaking, $\mathcal{E} \lfloor e$ is $\mathcal{E}$ minus the event e, and minus all events that are in conflict with e. The reachable LTS with initial state $\mathcal{E}$ corresponds to the computations over $\mathcal{E}$. It is usually defined using the notion of *configuration* [25]. However, by relying on the LTS as defined above, the adequacy theorem has a simpler formulation. A precise correspondence between the two notions of LTS can be easily defined.

Event structures have been shown to be the class of objects of a category [25], whose morphisms are defined as follows. Let $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1 \rangle$, $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2 \rangle$ be event structures. A *morphism* $f : \mathcal{E}_1 \to \mathcal{E}_2$ is a partial function $f : E_1 \to E_2$ such that

- f reflects causality: if $f(e_1)$ is defined, then $\lceil f(e_1) \rceil \subseteq f(\lceil e_1 \rceil)$;
- f reflects reflexive conflict: if $f(e_1), f(e_2)$ are defined, and if $f(e_1) \asymp f(e_2)$, then $e_1 \asymp e_2$.

It is easily shown that an isomorphism in this category is a bijective function that preserves and reflects causality and conflict. In the presence of labelled event structures $\mathcal{E}_1 = \langle E_1, \leq_1, \smile_1, \lambda_1 \rangle$, $\mathcal{E}_2 = \langle E_2, \leq_2, \smile_2, \lambda_2 \rangle$ on the same set of labels L, we will consider only *label preserving* isomorphisms, i.e. isomorphisms $f : \mathcal{E}_1 \to \mathcal{E}_2$ such that $\lambda_2(f(e_1)) = \lambda_1(e_1)$. If there is an isomorphism $f : \mathcal{E}_1 \to \mathcal{E}_2$, we say that $\mathcal{E}_1, \mathcal{E}_2$ are isomorphic, written $\mathcal{E}_1 \cong \mathcal{E}_2$.

3.2 Operators on event structures

We provide here an informal description of several operations on labelled event structures, that we are going to use in the next section. See [22] for more details.

- *Prefixing $a.\mathcal{E}$.* This operation adds to the event structure a new minimal element, labelled by a, below every other event in $\mathcal{E}$. Conflict, order, and labels of original elements remain the same as in $\mathcal{E}$.

– *Prefixed sum* $\sum_{i\in I} a_i.\mathcal{E}_i$. This is obtained as the disjoint union of copies of the event structures $a_i.\mathcal{E}_i$. The order relation of the new event structure is the disjoint union of the orders of $a_i.\mathcal{E}_i$ and the labelling function is the disjoint union of the labelling functions of $a_i.\mathcal{E}_i$. As for the conflict relation, we take the disjoint union of the conflicts appearing in $a_i.\mathcal{E}_i$ and we extend it by putting in conflict every pair of events belonging to two different copies of $a_i.\mathcal{E}_i$.
– *Restriction* (or *Hiding*) $\mathcal{E}\setminus X$ where $X \subseteq L$ is a set of labels. This is obtained by removing from E all events with label in X and all events that are above (i.e., causally depend on) one of those. On the remaining events, order, conflict and labelling are unchanged.
– *Relabelling* $\mathcal{E}[f]$ where L and L' are two sets of labels and $f : L \to L'$. This operation just consists in composing the labelling function λ of $\mathcal{E}$ with the function. The new event structure is labelled over L' and its labelling function is $f \circ \lambda$.

3.3 The parallel composition

The parallel composition of two event structures $\mathcal{E}_1$ and $\mathcal{E}_2$ gives a new event structure $\mathcal{E}'$ whose events model the parallel occurrence of events $e_1 \in E_1$ and $e_2 \in E_2$. In particular, when the labels of e_1 and e_2 match according to an underlying synchronisation model, $\mathcal{E}'$ records (with an event $e' \in E'$) that a synchronisation between e_1 and e_2 is possible, and deals with the causal effects of such a synchronisation.

The parallel composition is defined as the categorical product followed by restriction and relabelling [25]. Even if the categorical product is unique up to isomorphism, it can be explicitly constructed in different ways. We give a brief outline of one such construction [9, 19]. Let $\mathcal{E}_1 := \langle E_1, \leq_1, \smile_1\rangle$ and $\mathcal{E}_2 := \langle E_2, \leq_2, \smile_2\rangle$ be event structures. Let $E_i^* := E_i \uplus \{*\}$, where $*$ is a distinguished event. The categorical product is given by an event structure $\mathcal{E} = \langle E, \leq, \smile\rangle$ and two morphisms $\pi_i : \mathcal{E} \to \mathcal{E}_i$ (the projections). The elements of E are of the form (W, e_1, e_2) where W is a finite subset of E, and $e_i \in E_i^*$. Intuitively W is the enabling set of the event (W, e_1, e_2). The order $\leq$ is generated by $(W, e_1, e_2) \leq (W', e_1', e_2')$ iff $(W, e_1, e_2) \in W'$. The conflict relation $\smile$ is defined using the conflict relations of E_1, E_2. The projections are defined as $\pi_1(W, e_1, e_2) = e_1$ and $\pi_2(W, e_1, e_2) = e_2$. For event structures with labels in L, let be $L_* := L \uplus \{*\}$ where $*$ is a distinguished label. Then the labelling function of the product takes on the set $L_* \times L_*$, and we define $\lambda(W, e_1, e_2) = (\lambda_1^*(e_1), \lambda_2^*(e_2))$, where $\lambda_i^*(e_i) = \lambda_i(e_i)$ if $e_i \neq *$, and $\lambda_i^*(*) = *$.

The synchronisation model underlying the relabelling operation needed for parallel composition is formalised by the following notion of *synchronisation algebra* [25]. A synchronisation algebra S is a partial binary operation $\bullet_S$ defined on L_*. If α_i is the label of an event $e_i \in E_i$, then $\alpha_1 \bullet_S \alpha_2$ gives the label of the event $e' \in E'$ representing the synchronisation of e_1 and e_2. If the synchronisation algebra is not defined, the synchronisation event is given a distinguished label bad that indicates that this event is not allowed and should be deleted.

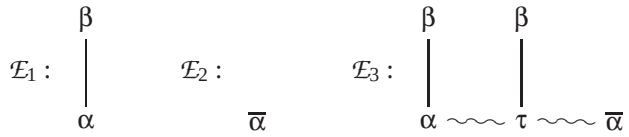
Definition 5 (Parallel Composition of Event Structures). *Let $\mathcal{E}_1, \mathcal{E}_2$ two event structures labelled over L, let S be a synchronisation algebra, and let $f_S : L_* \to L' = L_* \cup \{\mathsf{bad}\}$ be a function defined as $f_S(\alpha_1, \alpha_2) = \alpha_1 \bullet_S \alpha_2$, if S is defined on (α_1, α_2), and $f_S(\alpha_1, \alpha_2) = \mathsf{bad}$ otherwise. The parallel composition $\mathcal{E}_1 \|_S \mathcal{E}_2$ is defined as follows the*

categorical product followed by relabelling and restriction[1]*:*

$$\mathcal{E}_1 \|_S \mathcal{E}_2 = (\mathcal{E}_1 \times \mathcal{E}_2)[f_S] \setminus \{\mathsf{bad}\}$$

The subscripts S are omitted when the synchronisation algebra is clear from the context.

Example 2. We show a simple example of parallel composition. Let $L = \{\alpha, \beta, \overline{\alpha}, \tau\}$ Consider the two event structures $\mathcal{E}_1, \mathcal{E}_2$, where $E_1 = \{a, b\}, E_2 = \{a'\}$, with $a \leq_1 b$ and $\lambda_1(a) = \alpha, \lambda_1(b) = \beta, \lambda_2(a') = \overline{\alpha}$. The event structures are represented as follows:



where curly lines represent immediate conflict, while the causal order proceeds upwards along the straight lines. Consider the synchronisation algebra obtained as the symmetric closure of the following rules: $\alpha \bullet \overline{\alpha} = \tau$, $\alpha \bullet * = \alpha$, $\overline{\alpha} \bullet * = \overline{\alpha}$, $\beta \bullet * = \beta$ and undefined otherwise. Then $\mathcal{E}_3 := \mathcal{E}_1 \| \mathcal{E}_2$ is the event structure $\langle E_3, \leq, \smile, \lambda \rangle$ where $E_3 = \{e := (\emptyset, a, *), e' := (\emptyset, *, a'), e'' := (\emptyset, a, a'), d := (\{e\}, a', *), d'' := (\{e''\}, a', *)\}$, the ordering $\leq$ is defined as $e \leq d, e'' \leq d''$, while the conflict $\smile$ is defined as $e \smile e''$, $e' \smile e''$, $e \smile d''$, $e' \smile d''$, $e'' \smile d$, $d \smile d''$. The labelling function is $\lambda(e) = \alpha$, $\lambda(e') = \overline{\alpha}, \lambda(e'') = \tau, \lambda(d) = \lambda(d'') = \beta$.

3.4 A large CPO of event structures

We say that an event structure $\mathcal{E}$ is a *prefix* if an event structure $\mathcal{E}'$, denoted $\mathcal{E} \leq \mathcal{E}'$ if there exists $\mathcal{E}'' \cong \mathcal{E}'$ such that $E \subseteq E''$ and no event in $E'' \setminus E$ is below any event of E.

Winskel [21] has shown that the class of event structures with the prefix order is a large CPO, and thus the limits of countable increasing chains exist. Moreover all operators on event structures are continuous.

We will use this fact to define the semantics of the recursive definitions.

4 Event Structure Semantics

This section defines the denotational semantics of πI-processes by the labelled event structures. Given a process P, we associate to P an event structure $\mathcal{E}_P$ whose events e represent the occurrence of an action $\lambda(e)$ in the LTS of P. Moreover, our main issue is compositionality: the semantics of the process $P \mid Q$ should be defined as $\mathcal{E}_P \mid\mid \mathcal{E}_Q$ so that the operator $\mid\mid$ satisfactorily models the parallel composition of P and Q.

[1] The standard definition of parallel composition is $(\mathcal{E}_1 \times \mathcal{E}_2 \setminus X)[f]$, where the restriction and relabeling operations are swapped, and X is the set of labels (pairs) for which f is undefined. We can prove that such a definition is equivalent to ours, which is more suitable to be generalised to the π-calculus.

4.1 Generalised relabelling

It is clear form Definition 5 that the core of the parallel composition of event structures is the definition of a relabelling function encoding the intended synchronisation model. As discussed in the Introduction, name dependences appearing in πI-processes let a synchronisation between two events possibly depend on the previous synchronisations. We then define a generalised relabelling operation where the relabelling of an event depends on (the labels of) its causal history. Such a new operator is well-suited to encode the πI-communication model and allows the semantics of the πI-calculus to be defined as an extension of CCS event structure semantics.

Definition 6 (Generalised Relabelling). *Let L and L' be two sets of labels, and let $Pom(L')$ be a pomset (i.e., partially ordered multiset) of labels in L'. Given an event structure $\mathcal{E} = \langle E, \leq, \smile, \lambda \rangle$ over the set of labels L, and a function $f : Pom(L') \times L \longrightarrow L'$, we define the relabelling operation $\mathcal{E}[f]$ as the event structure $\mathcal{E}' = \langle E, \leq, \smile, \lambda' \rangle$ with labels in L', where $\lambda' : E \longrightarrow L'$ is defined as follows by induction on the height of an element of E:*

$$\text{if } h(e) = 0 \text{ then } \lambda'(e) = f(\emptyset, \lambda(e))$$
$$\text{if } h(e) = n+1 \text{ then } \lambda'(e) = f(\lambda'([e)), \lambda(e))$$

In words, an event e is relabelled with a label $\lambda'(e)$ that depends on the (pomset of) labels of the events belonging to its causal history $[e)$.

The set of labels we consider is $L = \{a(x), \overline{a}(x), \tau \mid a, x \in Names\}$. For the parallel composition we need an auxiliary set of labels $L' = \{a(x), \overline{a}(x), \tau_{x=y} \mid a, x, y \in Names\} \cup \{\mathsf{bad}, \mathsf{hide}\}$, where bad and hide are distinguished labels.

In L', the silent action τ is tagged with the couple of bound names that get identified through the synchronisation. This extra piece of information carried by τ-actions is essential in the definition of the generalised relabelling function. Let for instance e encode the parallel occurrence of two events e_1, e_2 labelled, resp., $x(x')$ and $\overline{y}(y')$, then e_1 and e_2 do synchronise only if x and y are equal, that is only if in the causal history of e there is an event labelled with $\tau_{x=y}$; in such a case e can then be labelled with $\tau_{x'=y'}$.

The distinguished label bad denotes, as before, synchronisations that are not allowed, while the new label hide denotes the hiding of newly generated names. Both labels are finally deleted.

Let $f_\pi : Pom(L') \times (L \uplus \{*\} \times L \uplus \{*\}) \longrightarrow L'$ be the relabelling function defined as:

$$f_\pi(X, \langle a(y), \overline{a}(z) \rangle) = f_\pi(X, \langle \overline{a}(z), a(y) \rangle) = \tau_{y=z}$$
$$f_\pi(X, \langle a(y), \overline{b}(z) \rangle) = f_\pi(X, \langle \overline{b}(z), a(y) \rangle) = \begin{cases} \tau_{y=z} & \text{if } \tau_{a=b} \in X \\ \mathsf{bad} & \text{otherwise} \end{cases}$$
$$f_\pi(X, \langle a(y), * \rangle) = f_\pi(X, \langle *, a(y) \rangle) = \begin{cases} \mathsf{hide} & \text{if } \tau_{a=b} \in X \\ a(y) & \text{otherwise} \end{cases}$$
$$f_\pi(X, \langle \overline{a}(y), * \rangle) = f_\pi(X, \langle *, \overline{a}(y) \rangle) = \begin{cases} \mathsf{hide} & \text{if } \tau_{a=b} \in X \\ \overline{a}(y) & \text{otherwise} \end{cases}$$
$$f_\pi(X, \langle \alpha, \beta \rangle) = \mathsf{bad} \quad \text{otherwise}$$

The function f_π encodes the πI-synchronisation model in that it only allows synchronisations between input and output over the same channel, or over two channels whose names have been identified by a previous communication. The actions over a channel a that has been the object of a previous synchronisation are relabelled as hide since, according to internal mobility, a is a bound name.

The extra information carried by the τ-actions is only necessary in order to *define* the relabelling, but it should later on be forgotten, as we do not distinguish τ-actions in the LTS. Hence we apply a second relabelling *er* that simply erases the tags:

$$er(\alpha) = \begin{cases} \tau & \text{if } \alpha = \tau_{x=y} \\ \alpha & \text{otherwise} \end{cases}$$

4.2 Definition of the semantics

The semantics of the πI-calculus is then defined as follows by induction on processes, where the parallel composition of event structure is defined by

$$\mathcal{E}_1 \|_\pi \mathcal{E}_2 = ((\mathcal{E}_1 \times \mathcal{E}_2)\ [f_\pi][er]) \setminus \{\mathsf{bad}, \mathsf{hide}\}$$

To deal with recursive definitions, we use an index k to denote the level of unfolding.

$$\{\!|\mathbf{0}|\!\}_k = \emptyset \qquad \{\!|\textstyle\sum_{i\in I} \pi_i.P_i|\!\}_k = \sum_{i\in I} \pi_i.\{\!|P_i|\!\}_k$$

$$\{\!|P \mid Q|\!\}_k = \{\!|P|\!\}_k \|_\pi \{\!|Q|\!\}_k \qquad \{\!|(\nu a)P|\!\}_k = \{\!|P|\!\}_k \backslash \{l \in L \mid a \text{ is the subject of } l\}$$

$$\{\!|A\langle \tilde{y} \mid \mathbf{w}\rangle|\!\}_0 = \emptyset \qquad \{\!|A\langle \tilde{y} \mid \mathbf{w}\rangle|\!\}_{k+1} = \{\!|P_A\{\tilde{y}/\tilde{x}\}\{\mathbf{w}/\mathbf{z}\}|\!\}_k$$

Recall that all operators on event structures are continuous with respect to the prefix order. It is thus easy to show that, for any k, $\{\!|P|\!\}_k \leq \{\!|P|\!\}_{k+1}$. We define $\{\!|P|\!\}$ to be the limit of the increasing chain $\ldots\{\!|P|\!\}_k \leq \{\!|P|\!\}_{k+1} \leq \{\!|P|\!\}_{k+2}\ldots$:

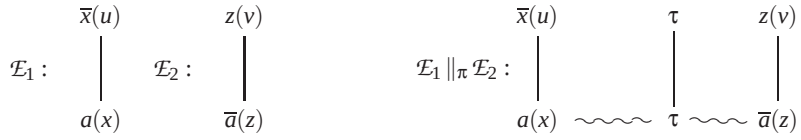
$$\{\!|P|\!\} = \sup_{k\in\mathbb{N}} \{\!|P|\!\}_k$$

Since all operators are continuous with respect to the prefix order we also have the following result:

Theorem 1 (Compositionality). *The semantics $\{\!|P|\!\}$ is compositional, i.e. $\{\!|P \mid Q|\!\} = \{\!|P|\!\} \|_\pi \{\!|Q|\!\}$, and so on for all other operators.*

4.3 Examples

Example 3. As the first example, consider the process $P = a(x).\overline{x}(u) \mid \overline{a}(z).z(v)$ discussed in the Introduction. We show in the following the two event structures $\mathcal{E}_1, \mathcal{E}_2$ associated to the basic threads, as well as the event structure corresponding to $\{\!|P|\!\} = \mathcal{E}_1 \|_\pi \mathcal{E}_2$. Figure 1 shows two intermediate steps involved in the construction of $\{\!|P|\!\}$, according to the definition of the parallel composition operator.



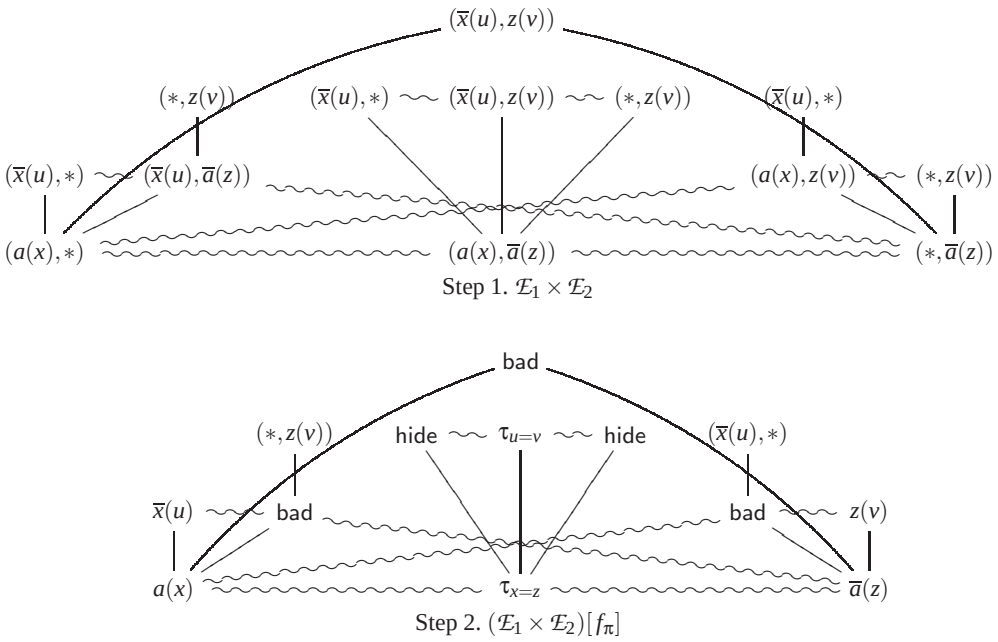
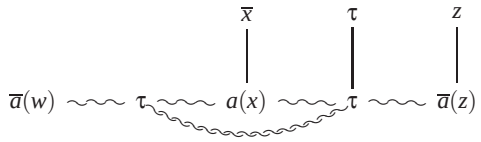
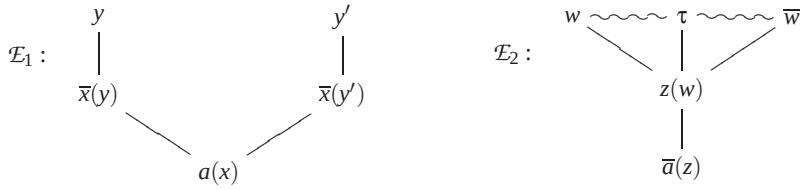


Fig. 1. Event structure corresponding to $a(x).\overline{x}(u) \mid \overline{a}(z).z(v)$

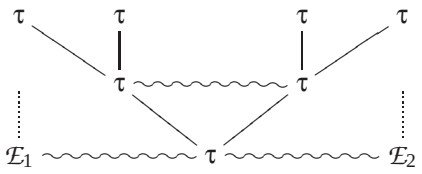
Example 4. As the second example, consider $Q = \overline{a}(w) \mid P$, where P is the process above. In Q two different communications may take place along the channel a: either the fresh name w is sent, and the resulting process is stuck, or the two threads in P can synchronise as before establishing a private channel for a subsequent communication. The behaviour of Q is illustrated by the following event structure which corresponds to $\{\!|\, Q \,|\!\} = \mathcal{E}_3 \|_\pi \{\!|\, P \,|\!\}$, where $\mathcal{E}_3 = \{\!|\, \overline{a}(w) \,|\!\}$ is a simple event structure consisting of a single event labeled by $\overline{a}(w)$.



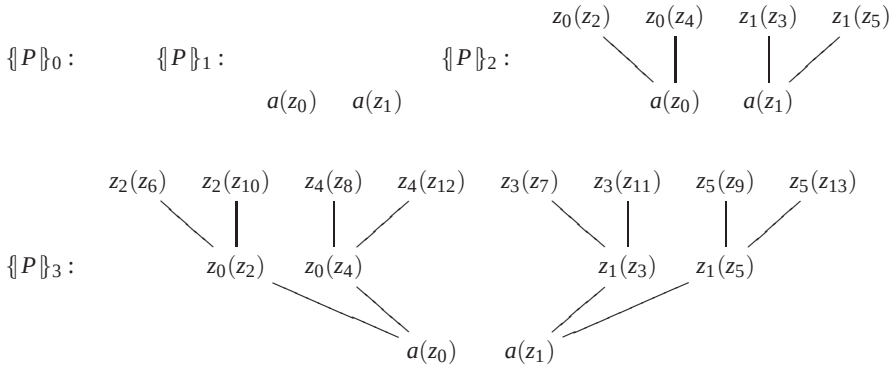
Example 5. As a further example, let $R = a(x).\big(\overline{x}(y).y \mid \overline{x}(y').y'\big) \mid \overline{a}(z).\big(z(w).(w \mid \overline{w})\big)$ whose two threads correspond to the following two event structures:



R allows a first communication on a that identifies x and z and triggers a further synchronisation with one of the outputs over x belonging to $\mathcal{E}_1$. This second communication identifies w with either y or y', which can now compete with w for the third synchronisation. The event structure corresponding to $\{\!|R|\!\} = \mathcal{E}_1 \|_\pi \mathcal{E}_2$ is the following; its construction is shown in Appendix B.



Example 6. Consider the recursive process, seen in Example 1 in Section 2, $A(x \mid \mathbf{z}) = x(z_0).A\langle z_0 \mid \mathbf{z}'\rangle \mid x(z_1).A\langle z_1 \mid \mathbf{z}''\rangle$, where $\mathbf{z}'(n) = \mathbf{z}(2n+2)$ and $\mathbf{z}''(n) = \mathbf{z}(2n+3)$. In the following, we draw the first approximations of the semantics of $P = A\langle a \mid \mathbf{z}\rangle$:



4.4 Properties of the semantics

The operational correspondence is stated in terms of the labelled transition system defined in Section 3.

Theorem 2 (Operational Adequacy). *Suppose* $P \xrightarrow{\beta} P'$ *in the* π*-calculus. Then* $\{\!|P|\!\} \xrightarrow{\beta} \cong \{\!|P'|\!\}$. *Conversely, suppose* $\{\!|P|\!\} \xrightarrow{\beta} \mathcal{E}'$. *Then there exists* P' *such that* $P \xrightarrow{\beta} P'$ *and* $\{\!|P'|\!\} \cong \mathcal{E}'$.

The proof technique is similar to the one used in [19], but it takes into account the generalised relabelling.

As an easy corollary, we get that the semantics is sound with respect to bisimilarity.

Theorem 3 (Soundness). *If* $\{\!|P|\!\} \cong \{\!|Q|\!\}$, *then* $P \sim Q$.

The converse of the soundness theorem (i.e. completeness) does not hold. In fact this is always the case for event structure semantics (for instance the one in [21]), because

bisimilarity abstracts away from causal relations, which are instead apparent in the event structures. As a counterexample, we have $a.b+b.a \sim a \mid b$ but $\{\!|\, a.b+b.a \,|\!\} \not\cong \{\!|\, a \mid b \,|\!\}$.

Isomorphism of event structures is indeed a very fine equivalence, however it is, in a sense behavioural, as it is *strictly* coarser than structural congruence.

Proposition 2. *If $P \equiv Q$ then $\{\!|\, P \,|\!\} \cong \{\!|\, Q \,|\!\}$*

The converse of the previous proposition does not hold: $\{\!|\, (\nu a)a.P \,|\!\} \cong \{\!|\, \mathbf{0} \,|\!\} = \emptyset$ but $(\nu a)a.P \not\equiv \mathbf{0}$. As a further counterexample, we have $(\nu a)(a(x).\overline{x}(u) \mid \overline{a}(y).y(v)) \not\equiv (\nu a,b)(a(x).\overline{b}(u) \mid \overline{a}(y).b(v))$, but both processes correspond to the same event structure containing only two events e_1, e_2 with $e_1 \leq e_2$ and $\lambda(e_1) = \lambda(e_2) = \tau$.

5 Asynchronous πI-calculus

This section studies the *asynchronous* πI-calculus [4, 14, 16], whose syntax slightly differs from that in Section 2 in the treatment of the output.

$$\textit{Processes } P,Q ::= \textstyle\sum_{i\in I} a_i(x_i).P_i \mid \overline{a}(x)P \mid P \mid Q \mid (\nu a)P \mid A\langle \tilde{x} \mid \mathbf{z} \rangle$$
$$\textit{Definition} \quad A(\tilde{x} \mid \mathbf{z}) = P_A$$

The new syntax of the bound output reflects the fact that there is a looser causal connection between the output and its continuation. A process $\overline{a}(x)P$ is different from $\overline{a}(x).P$ in that it can activate the process P even if the name x has not been emitted yet along the channel a. The operational semantics can be obtained from that of Section 2 by removing the rule (OUT) and adding the following three rules:

$$\text{(OUT)}\ \frac{}{\overline{a}(x)P \xrightarrow{\overline{a}(x)} P} \qquad \text{(ASYNC)}\ \frac{P \xrightarrow{\alpha} P'}{\overline{a}(x)P \xrightarrow{\alpha} \overline{a}(x)P'}\ x \notin \mathrm{fn}(\alpha) \qquad \text{(ASYNCH COMM)}\ \frac{P \xrightarrow{a(y)} P'}{\overline{a}(x)P \xrightarrow{\tau} (\nu x)P'\{x/y\}}$$

Relying on this LTS, the definition of strong bisimilarity for the asynchronous πI-calculus is identical to that in Section 2.

5.1 Denotational semantics

The event structure semantics of the asynchronous πI-calculus requires to encode the output process $\overline{a}(x)P$, introducing the following novel operator, called *rooting*.

Definition 7 (Rooting $a[X].\mathcal{E}$). *Let $\mathcal{E}$ be an event structure labelled over L, let a be a label and $X \subseteq L$ be a set of labels. We define the rooting operation $a[X].\mathcal{E}$ as the event structure $\mathcal{E}' = \langle E', \leq', \smile', \lambda' \rangle$, where $E' = E \uplus \{e'\}$ for some new event e', $\leq'$ coincides with $\leq$ on E and for every $e \in E$ such that $\lambda(e) \in X$ we have $e' \leq' e$, the conflict relation $\smile'$ coincides with $\smile$, that is e' is in conflict with no event. Finally, λ' coincides with λ on E and $\lambda'(e') = a$.*

The rooting operation adds to the event structure a new event, labeled by a, which is put below the events with labels in X (and any event above them).

The rooting operation is used to give the semantics of asynchronous bound output. Given a process $\overline{a}(x)P$, every action performed by P that has x as subject is rooted with a distinctive label $\bot$. The resulting structure is composed in parallel with $\overline{a}(x)$, so that (*i*) every action that does not depend on x can synchronise with $\overline{a}(x)$, and (*ii*) the actions rooted by $\bot$ (i.e. those depending on x) become causally dependent on the action $\overline{a}(x)$.

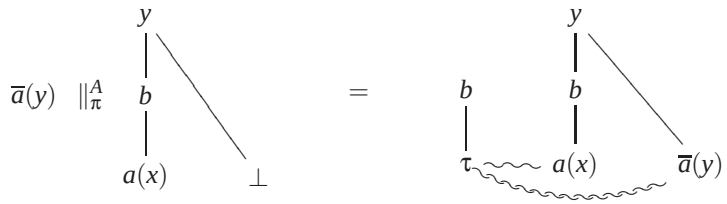
Such a composition is formalised the parallel composition operator $\|^A_\pi$ built around the generalised relabelling function $f^A_\pi : Pom(L') \times (L \uplus \{*, \bot\} \times L \uplus \{*, \bot\}) \longrightarrow L'$ that extends f_π with the following two clauses dealing with the new labels:

$$f^A_\pi(X, \langle \bot, \overline{a}(x) \rangle) = f^A_\pi(X, \langle \overline{a}(x), \bot \rangle) = \overline{a}(x)$$
$$f^A_\pi(X, \langle \bot, * \rangle) = \quad f^A_\pi(X, \langle *, \bot \rangle) = \mathsf{bad}$$

The denotational semantics of asynchronous πI-processes is then identical to that in Section 4, with a new construction for the output:

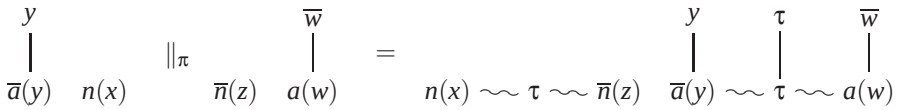
$$\{\!|\, \overline{a}(x)P \,|\!\}_k = \overline{a}(x) \;\|^A_\pi\; \bot[X].\{\!|\, P \,|\!\}_k \qquad X = \{\alpha \in L \mid x \text{ is the subject of } \alpha\}$$

Example 7. Let R be the process $\overline{a}(y)(a(x).b.y)$; its semantics is defined by the following event structure:



First a new event labelled by $\bot$ is added below any event whose label has y as subject. In this case there is only one such event, labelled by y. Then the resulting event structure is put in parallel with the single event labelled by $\overline{a}(y)$. This event can synchronise with the $\bot$ event or with the $a(x)$ event. The first synchronisation simply substitutes the label $\overline{a}(y)$ for $\bot$. The second one behaves as a standard synchronisation.

Example 8. The semantics of the process $P = \overline{a}(y)(n(x) \mid y) \;\mid\; \overline{n}(z)(a(w).\overline{w})$ discussed above is the following event structure:



Note that the causality between the $a(w)$ event and the $\overline{w}$ event is both object *and* subject, and it is due to the prefix constructor. The causality between the $\overline{a}(y)$ event and the y event is only object, and it is due to the rooting.

5.2 Properties of the semantics

As for the synchronous case, the semantics is adequate with respect to the labelled transition system.

Theorem 4 (Operational Adequacy). *Suppose* $P \xrightarrow{\beta} P'$ *in the* π*-calculus. Then* $\{\!|P|\!\} \xrightarrow{\beta} \cong \{\!|P'|\!\}$. *Conversely, suppose* $\{\!|P|\!\} \xrightarrow{\beta} \mathcal{E}'$. *Then there exists* P' *such that* $P \xrightarrow{\beta} P'$ *and* $\{\!|P'|\!\} \cong \mathcal{E}'$.

The proof is analogous to the synchronous case, with a case analysis for the rooting.

Theorem 5 (Soundness). *If* $\{\!|P|\!\} \cong \{\!|Q|\!\}$, *then* $P \sim Q$.

6 Related and Future Work

There are several causal models for the π-calculus, that use different techniques. There exist semantics in terms of labelled transition systems, where the causal relations between transitions are represented by "proofs" which allow to distinguish different occurrences of the same transition [3, 10]. In [7], a more abstract approach is followed, which involves indexed transition systems. In [15], a semantics of the π-calculus in terms of pomsets is given, following ideas from dataflow theory. The two papers [6, 11] present Petri nets semantics of the π-calculus.

A direct antecedent of this work presented a compositional, sound and adequate event structure semantics for a restricted, typed variant of the π-calculus [19]. This subcalculus can embed the λ-calculus fully abstractly [1], but is strictly less expressive than the full π-calculus. The newly generated names of this subcalculus can be statically determined when typing processes, therefore the semantics presented there uses the original formulation of the parallel composition for CCS. The generalised relabelling, the rooting operator as well as the treatment of recursive definitions are developed first in the present paper.

A recent work [5] provides an event structure semantics of the π-calculus. However this semantics does not correspond to the labelled transition semantics, but only to the *reduction* semantics, i.e. only internal silent transitions are represented in the event structure. For instance, in [5], the processes $a(x)$ and $\mathbf{0}$ have both the same semantics, the empty event structure. Consequently the semantics is neither compositional, operationally adequate, nor an extension of Winskel's semantics of CCS.

Recently Winskel [24] used event structures in a different way to give semantics to a kind of value passing CCS. His recent work [23] extends the framework of [24] to a functor category that can handle new name generation, but does not apply yet to the π-calculus.

The close relation between concurrent game semantics, linear logic and event structure semantics of the typed π-calculus has already been observed in [19, 13, 12]. In both worlds, the types play an important role to restrict the amount of concurrency and nondeterminism. Based on the present work, it will be interesting to extend the relation to the untyped, fully non-deterministic and concurrent framework.

Our semantics captures the essential features of the causal dependencies created by both synchronous and asynchronous name passing. For an extension of free name passing, we plan to use a technique analogous to the one developed for the asynchronous πI-calculus. As observed in [3, 10], the presence of free outputs allows subtle forms of name dependences, as exemplified by $(\nu b)(\overline{a}\langle b\rangle \mid \overline{c}\langle b\rangle)$, where a restriction contributes

the object causality. A refinement of the rooting operator would be used for uniform handling name causalities induced by both internal and external mobility.

References

1. M. Berger, K. Honda, and N. Yoshida. Sequentiality and the π-calculus. In *TLCA'01*, volume 2044 of *LNCS*, pages 29–45. Springer, 2001.
2. M. Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theor. Comp. Sci.*, 195(2):205–226, 1998.
3. M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the π-calculus. *Acta Inf.*, 35(5):353–400, 1998.
4. G. Boudol. Asynchrony and the π-calculus. Research Report 1702, INRIA, 1992.
5. R. Bruni, H. Melgratti, and U. Montanari. Event structure semantics for nominal calculi. In *CONCUR'06*, volume 4137 of *LNCS*, pages 295–309. Springer, 2006.
6. N. Busi and R. Gorrieri. A petri net semantics for pi-calculus. In *CONCUR'95*, volume 962 of *LNCS*, pages 145–159. Springer, 1995.
7. G. L. Cattani and P. Sewell. Models for name-passing processes: Interleaving and causal. In *LICS'00*, pages 322–332. IEEE, 2000.
8. P.-L. Curien and C. Faggian. L-nets, strategies and proof-nets. In *CSL'05*, volume 3634 of *LNCS*, pages 167–183. Springer, 2005.
9. P. Degano, R. De Nicola, and U. Montanari. On the consistency of "truly concurrent" operational and denotational semantics. In *LICS'88*, pages 133–141. IEEE, 1988.
10. P. Degano and C. Priami. Non-interleaving semantics for mobile processes. *Theor. Comp. Sci.*, 216(1-2):237–270, 1999.
11. J. Engelfriet. A multiset semantics for the pi-calculus with replication. *Theor. Comp. Sci.*, 153(1&2):65–94, 1996.
12. C. Faggian and M. Piccolo. A graph abstract machine describing event structure composition. In *GT-VC workshop*, ENTCS, 2007.
13. C. Faggian and M. Piccolo. Ludics is a model for the (finitary) linear pi-calculus. In *TLCA'07*, LNCS. Springer, 2007.
14. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *ECOOP'91*, volume 512 of *LNCS*, pages 133–147. Springer, 1991.
15. L. J. Jagadeesan and R. Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus. In *AMAST'95*, volume 936 of *LNCS*, pages 277–291, 1995.
16. M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. *Math.Struc.Comp.Sci.*, 14:715–767, 2004.
17. M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theor. Comp. Sci.*, 13(1):85–108, 1981.
18. D. Sangiorgi. π-calculus, internal mobility and agent passing calculi. *Theor. Comp. Sci.*, 167(2):235–271, 1996.
19. D. Varacca and N. Yoshida. Typed event structures and the π-calculus. In *MFPS'06*, ENTCS, 2006. Full version available at `www.pps.jussieu.fr/~varacca`.
20. D. Varacca and N. Yoshida. The Probabilistic π-Calculus and Event Structures. In *QAPL'07*, ENTCS, 2007.
21. G. Winskel. Event structure semantics for CCS and related languages. In *ICALP'82*, volume 140 of *LNCS*, pages 561–576. Springer, 1982.
22. G. Winskel. Event structures. In *Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course*, volume 255 of *LNCS*, pages 325–392. Springer, 1987.
23. G. Winskel. Name generation and linearity. In *LICS'05*, pages 301–310. IEEE, 2005.
24. G. Winskel. Relations in concurrency. In *LICS'05*, pages 2–11. IEEE, 2005.
25. G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of logic in Computer Science*, volume 4. Clarendon Press, 1995.

A Appendix: Proofs

A.1 Definition of the substitution of sequences

Let $A(\tilde{x} \mid \mathbf{z}) = P_A$ be a recursive definition. We have that the sequence $\mathbf{z}$ contains all bound names of P_A. In particular the sequence $\mathbf{z}$ contains all names of all sequences $\mathbf{z}'$ that appear in P_A. Since all bound names are different, there exists an injective function $f : \mathbb{N} \to \mathbb{N}$ such that $\mathbf{z}'(n) = \mathbf{z}(f(n))$. Also all bound names of P_A are of the form $\mathbf{z}(n)$ for some n. The process $P_A\{\mathbf{w}/\mathbf{z}\}$ is defined as follows. For each bound name of the form $\mathbf{z}(n)$ we substitute $\mathbf{w}(n)$, and for each sequence $\mathbf{z}'$ we substitute the sequence $\mathbf{w}'$ defined as $\mathbf{w}'(n) = \mathbf{w}(f(n))$.

Formally:

- $(a(x).P)\{\mathbf{w}/\mathbf{z}\} = a(y).P\{\mathbf{w}'/\mathbf{z}'\}$, where $x = \mathbf{z}(n)$, $y = \mathbf{w}(n)$, for some n; $\mathbf{z}'(k) = \mathbf{z}(k)$ if $k < n$ and $\mathbf{z}'(k) = \mathbf{z}(k+1)$ otherwise; $\mathbf{w}'(k) = \mathbf{w}(k)$ if $k < n$ and $\mathbf{w}'(k) = \mathbf{w}(k+1)$ otherwise.
- $(P \mid Q)\{\mathbf{w}/\mathbf{z}\} = P\{\mathbf{w}'/\mathbf{z}'\} \mid Q\{\mathbf{w}''/\mathbf{z}''\}$ where the sequences $\mathbf{z}', \mathbf{z}''$ partition the sequence $\mathbf{z}$, i.e. there exist two injection function $f, g : \mathbb{N} \to \mathbb{N}$ such that $f(\mathbb{N}) \cap g(\mathbb{N}) = \emptyset$, $f(\mathbb{N}) \cup g(\mathbb{N}) = \mathbb{N}$ and $\mathbf{z}'(n) = \mathbf{z}(f(n))$ and $\mathbf{z}''(n) = \mathbf{z}(g(n))$. Then $\mathbf{w}', \mathbf{w}''$ are defined by $\mathbf{w}'(n) = \mathbf{w}(f(n))$ and $\mathbf{w}''(n) = \mathbf{w}(g(n))$.
- $((\nu a)P)\{\mathbf{w}/\mathbf{z}\}$ is defined as for the prefix.
- $A\langle \tilde{x} \mid \mathbf{z} \rangle\{\mathbf{w}/\mathbf{z}\} = A\langle \tilde{x} \mid \mathbf{w} \rangle$.

A.2 Proof of Theorem 2

The proof is by induction on the rules of the operational semantics. All cases are rather straightforward, except the parallel composition and recursion (and output for the asynchronous calculus). For the parallel composition the crucial lemma is the following, where $y \mapsto x$ is the relabelling function $f(\alpha) = \alpha\{x/y\}$ and X is the set of labels with x as subject.

Lemma 1. *Let $\cong$ denote isomorphism of event structures. Let $X \subseteq L$ be the set of labels with subject x. We have that $\mathcal{E}_1 \xrightarrow{\overline{a}(x)} \mathcal{E}_1 \lfloor e_1$, and $\mathcal{E}_2 \xrightarrow{a(y)} \mathcal{E}_2 \lfloor e_2$ if and only if $\mathcal{E}_1 \| \mathcal{E}_2 \xrightarrow{\tau} \mathcal{E}_1 \| \mathcal{E}_2 \lfloor (\emptyset, e_1, e_2)$. Moreover, in such a case, we have $\mathcal{E}_1 \| \mathcal{E}_2 \lfloor (\emptyset, e_1, e_2) \cong ((\mathcal{E}_1 \lfloor e_1) \| (\mathcal{E}_2 \lfloor e_2 [y \mapsto x])) \setminus X$.*

The first part of theorem is straightforward: if e_1, e_2 are minimal in $\mathcal{E}_1, \mathcal{E}_2$, then $(\emptyset, e_1, e_2)$ is a minimal event in $\mathcal{E}_1 \| \mathcal{E}_2$, and vice versa. Assuming this is the case, one can prove that $\mathcal{E}_1 \| \mathcal{E}_2 \lfloor (\emptyset, e_1, e_2) \cong (\mathcal{E}_1 \lfloor e_1) \| (\mathcal{E}_2 \lfloor e_2)$. This is done by defining a bijective function $f : \mathcal{E}_1 \| \mathcal{E}_2 \lfloor (\emptyset, e_1, e_2) \to ((\mathcal{E}_1 \lfloor e_1) \| (\mathcal{E}_2 \lfloor e_2 [y \mapsto x])) \setminus X$ such that both f and f^{-1} are morphism of event structures. The definition of f and the proofs that it is a isomorphism is similar to the one found in [20].

A similar lemma is used for the adequacy with respect to the asynchronous prefix.

Lemma 2. *Let $\mathcal{E}' = \overline{a}(x) \parallel_\pi^A \perp[X].\mathcal{E}$. Then we have that*

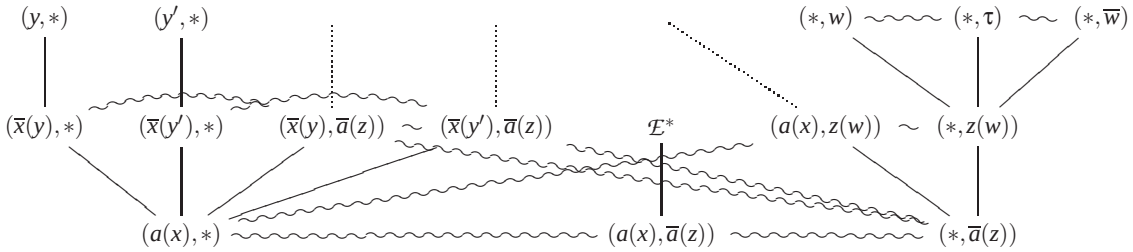
- $\mathcal{E}' \xrightarrow{\overline{a}(x)} \cong \mathcal{E}$
- $\mathcal{E} \xrightarrow{\beta} \mathcal{E} \lfloor e$ *where x is not the subject of β, if and only if $\mathcal{E}' \xrightarrow{\beta} \cong \overline{a}(x) \parallel_\pi^A \perp[X].\mathcal{E} \lfloor e$*

– $\mathcal{E} \xrightarrow{a(y)} \cong \mathcal{E}\lfloor e$, *if and only if* $\mathcal{E}' \xrightarrow{\tau} \mathcal{E}\lfloor e[y \mapsto x] \setminus X$

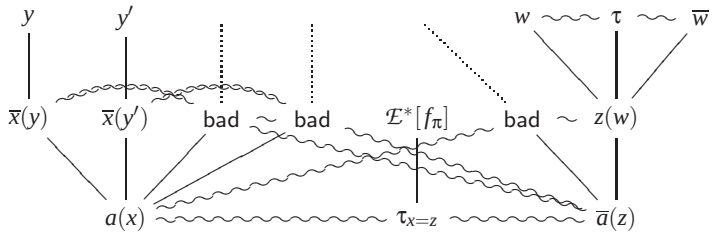
Finally, for the recursion, it is enough to observe that any minimal event of the denotation of a recursive process must belong to all but finitely many approximations.

B Appendix: Examples from Section 4

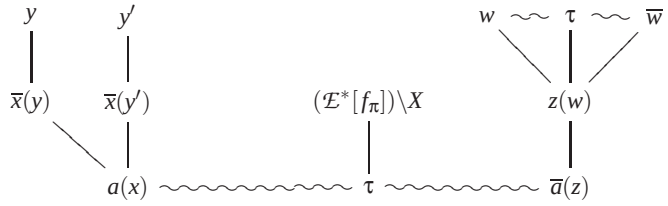
We show the construction of the examples of parallel composition.



Step 1. $\mathcal{E}_1 \times \mathcal{E}_2$, where the definition of $\mathcal{E}^*$ is in Figure 2

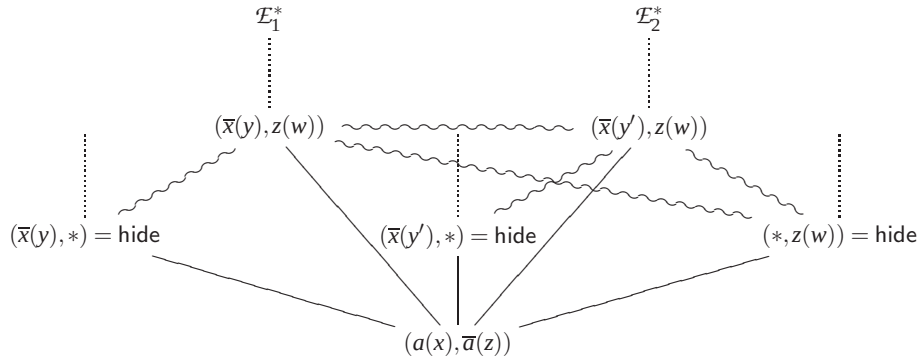


Step 2. $(\mathcal{E}_1 \times \mathcal{E}_2)[f_\pi]$, where the definition of $\mathcal{E}^*[f_\pi]$ is in Figure 2

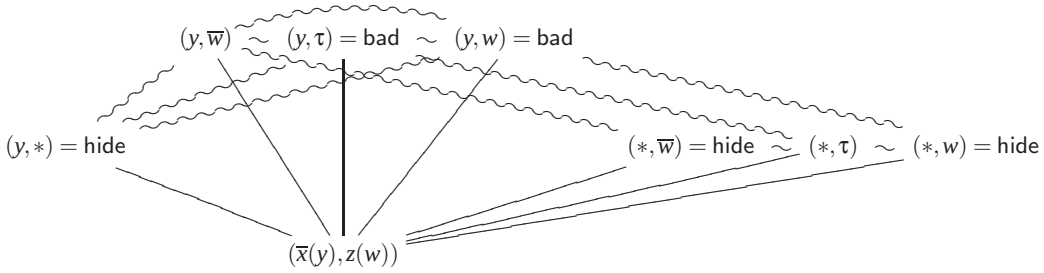


Step 3. $\mathcal{E}_1 \,||\, \mathcal{E}_2$ where the definition of $\mathcal{E}^*[f_\pi][er]\backslash X$ is in Figure 2

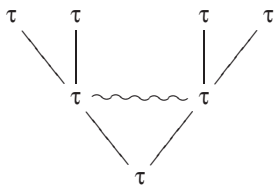
Table 1. Event structure corresponding to $\{\!|R|\!\}$



Definition of $\mathcal{E}^*$, where $\mathcal{E}_1^*$ is shown below and $\mathcal{E}_2^*$ is identical to $\mathcal{E}_1^*$ where y has been substituted with y'.



Definition of $\mathcal{E}_1^*$.



Definition of $(\mathcal{E}^*[f_\pi][er])\backslash X$

Table 2. Event structure corresponding to $\{\!|R|\!\}$